3-dimensional binaural sound rendering using acoustic measurements stored in the SOFA format

Nicolás Caputo Albacete

Institut für Nachrichtentechnik
Universität Rostock

April 2013

PROYECTO FIN DE CARRERA

TEMA: Renderizado binaural de audio en 3D.

TÍTULO: 3-dimensional binaural sound rendering using acoustics measurements stored in the SOFA format.

AUTOR: Nicolás Caputo Albacete

TITULACIÓN: Ingeniería Técnica de Telecomunicaciones. Especialidad en Imagen y Sonido

TUTOR: Prof. Dr.-Ing. Sascha Spors

CENTRO DE LECTURA: Institut für Nachrichtentechnik. Universität Rostock

FECHA DE LECTURA: 16 de abril de 2013

Resumen

El SSR (SoundScape Renderer) es un programa que está siendo desarrollado actualmente por la Universität Rostock, y previamente por la Technische Universität Berlin. El SSR es una herramienta diseñada para la reproducción y renderización de audio 2D en tiempo real. Para ello utiliza diversos algoritmos, algunos orientados a sistemas formados por arrays de altavoces en diferentes configuraciones y otros algoritmos diseñados para cascos. El principal objetivo de este proyecto es dotar al SSR de la capacidad de renderizar sonidos binaurales en 3D.

Este proyecto está centrado en el binaural renderer del SSR. Este algoritmo se basa en el uso de HRTFs (Head Related Transfer Function). Las HRTFs representan la función de transferencia del sistema formado por la cabeza y el torso del oyente. Esta función es medida desde diferentes angulos. Con estos datos el binaural renderer puede generar audio en tiempo real simulando la posición de diferentes fuentes.

Para poder incluir una base de datos con HRTFs en 3D se ha hecho uso del nuevo formato SOFA (Spatially Oriented Format for Acoustics). Este nuevo formato se encuentra en una fase bastante temprana de su desarrollo. Está pensado para servir como formato estándar para almacenar HRTFs y cualquier otro tipo de medidas acústicas, ya que actualmente cada laboratorio cuenta con su propio formato de almacenamiento y esto hace bastante difícil usar varias bases de datos diferentes en un mismo proyecto.

El formato SOFA hace uso del contenedor numérico netCDF, que a su vez esta basado en un contenedor más básico llamado HRTF-5.

Para poder incluir el formato SOFA en el binaural renderer del SSR se ha desarrollado una API en C++ para poder crear y leer archivos SOFA con el fin de utilizar los datos contenidos en ellos dentro del SSR.

Abstract

Several groups all over the world are researching in several ways to render 3D sounds. One way to achieve this is to use Head Related Transfer Functions (HRTFs). These measurements contain the Frequency Response of the human head and torso for each angle. Some years ago, was only possible to measure these Frequency Responses only in the horizontal plane. Nowadays, several improvements have made possible to measure and use 3D data for this purpose.

The problem was that the groups didn't have a standard format file to store the data. That was a problem when a third part wanted to use some different HRTFs for 3D audio rendering. Every of them have different ways to store the data. The Spatially Oriented Format for Acoustics or SOFA was created to provide a solution to this problem. It is a format definition to unify all the previous different ways of storing any kind of acoustics data. At the moment of this project they have defined some basis for the format and some recommendations to store HRTFs. It is actually under development, so several changes could come.

The SOFA[1] file format uses a numeric container called netCDF[2], specifically the Enhaced data model described in netCDF 4 that is based on HDF5[3].

The SoundScape Renderer (SSR) is a tool for real-time spatial audio reproduction providing a variety of rendering algorithms. The SSR was developed at the Quality and Usability Lab at TU Berlin and is now further developed at the Institut für Nachrichtentechnik at Universität Rostock [4].

This project is intended to be an introduction to the use of SOFA files, providing a C++ API to manipulate them and adapt the binaural renderer of the SSR for working with the SOFA format.

Table of contents

Table of figures

Table of tables

1. Introduction

Several groups all over the world are researching in several ways to render 3D sounds. One way to achieve this is to use Head Related Transfer Functions (HRTFs). These measurements contains the Frequency Response of the human head and torso for each angle. Some years ago, was only possible to measure these Frequency Responses only in the horizontal plane. Nowadays, several improvements have made possible to measure and use 3D data for this purpose.

The problem was that the groups didn't have a standard format file to store the data. That was a problem when a third part wanted to use some different HRTFs for 3D audio rendering. Every of them have different ways to store the data. The Spatially Oriented Format for Acoustics or SOFA was created to provide a solution to this problem. It is a format definition to unify all the previous different ways of storing any kind of acoustics data. At the moment of this project they have defined some basis for the format and some recommendations to store HRTFs. It is actually under development, so several changes could come.

The SOFA[1] file format uses a numeric container called netCDF[2], specifically the Enhaced data model described in netCDF 4 that is based on HDF5[3]. More specific definition of SOFA, netCDF 4, and HDF5 will be given in chapters 5 and 6.

The SoundScape Renderer (SSR) is a tool for real-time spatial audio reproduction providing a variety of rendering algorithms. The SSR was developed at the Quality and Usability Lab at TU Berlin and is now further developed at the Institut für Nachrichtentechnik at Universität Rostock [4]. More details about SSR will be given in chapter 4.

This project is intended to be an introduction to the use of SOFA files, providing a C++ API to manipulate them and adapt the binaural renderer of the SSR for working with the SOFA format.

2. State of the Art

2.1 HRTFs explanation

A HRTF is the frequency response of the linear system formed by the human head, torso and pinna. It defines how the sound arrives to the human ears from a specific point of the space. The frequency response of a linear system is the ratio between the frequency spectrum of the output and the input. In the case of HRTFs the system is the human body, specifically the head, pinna and the torso.

The most typical set-up used to measure HRTFs is in a anechoic room, with the subject placed in the centre of the room and around him/her a system with one or more loudspeakers that can be moved to measure all the directions. Each research group have their own set-up, but the main idea is the same. One common difference can be that sometimes the loudspeakers are in a fixed position, and the person is rotated to record all the positions.

To record the signals that come from the loudspeaker 2 microphones are placed inside the ears. Then to obtain the HRTFs the signals are processed. HRTFs are frequency responses, so to obtain them is necessary to obtain the FFT of the input and output signals. The input signal is the one that is in the loudspeaker and the output is the recorded by the microphones.

Other method to measure HRTFs is to generate an approximation of a Dirac impulse with the loudspeakers, the signal recorded by the microphones would be Head Related Impulse Response (HRIRs). The HRTF is only the Fourier transform of HRIR.

The HRTFs are different for each person, because the form of the pinna, torso, head, … are different. Sometimes to measure them dummy heads are used to try to make them close enough to a wide range of people.

As presented before, several groups all over the world have public databases of HRTFs. The problem is that all of them have different ways to store the measurements. These are the databases supported by SOFA:

- CIPIC [5]
- LISTEN [6]
- T-Labs [7]
- MARL-NYU [8]
- Nagoya [9]
- ARI Database [10]

Most of the databases use matlab to store and manipulate the data, but inside these files the structures and fields are completely different.

3. Main goals

The main goal of this project is to provide to the SSR, and more specifically to the binaural renderer, the capability of generate 3D spatial sound using the new file format SOFA. To achieve the main goal there are some steps before.

Create some programs in C++ to read and write SOFA files. There is an API provided by the SOFA team for Matlab and Octave[11], but there is not for C++. It is necessary for using the data stored in a SOFA file to read it with the numeric container netCDF.

Obtain a SOFA file with 3D HRTFs stored on it. For this the examples from the SOFA web page can be used[11]. These examples use ARI database and T-Labs HRTFs.

Find a way to choose the correct HRIR from the file when the position of the source is specified with 2 angles. The resolution of the different measurements from different institutions may vary, so it is necessary to choose the nearest measurement to the given position.

Create a test program in C++ for testing. A very simple program that convolves some audio signal with the HRTFs read from the sofa to test that the reading from the file is correct.

Introduce the 3D binaural renderer in the source code of the SSR. The current GUI is not for 3D so an automatic method to move the sound sources over the audio scene will be necessary, or develop an alternative way to move them.

4. SSR description

4.1 General description

The SoundScape Renderer (SSR) is a tool for real-time spatial audio reproduction providing a variety of rendering algorithms. The SSR was developed at the Quality and Usability Lab at TU Berlin and is now further developed at the Institut für Nachrichtentechnik at Universität Rostock. It is intended as a tool to contribute to the spatial audio research.

The SSR has many different renderers. Some of them are oriented to headphones and others to loudspeakers arrays[12].

4.2 Binaural renderer description

This project is only focused in the binaural renderer. In this renderer there are some audio sources that can be moved around one single listener. The listener is static and only can be rotated. Once some sources are placed in the scene, the SSR calculates the angle between the listener and each source. Then reads the closest HRTFs and convolves it with the sound signal from the source. One problem here is that the HRTFs databases have a limited spatial resolution, and sometimes the sound rendered comes from a slightly different direction that the specified in the GUI.

The binaural renderer is not the only one which uses HRIRs, also the Binaural Room Synthesis Renderer (BRSR) makes use of them. In this case the renderer uses one dedicated set oh HRIRs for each source, called Binaural Room Impulse Response (BRIRs). The SOFA format allows to store this kind of acoustic data, so the BRSR could use also this format.

Nowadays the SSR only support two-dimensional rendering, but the goal of this project is to provide three-dimensional reproduction for the binaural renderer.

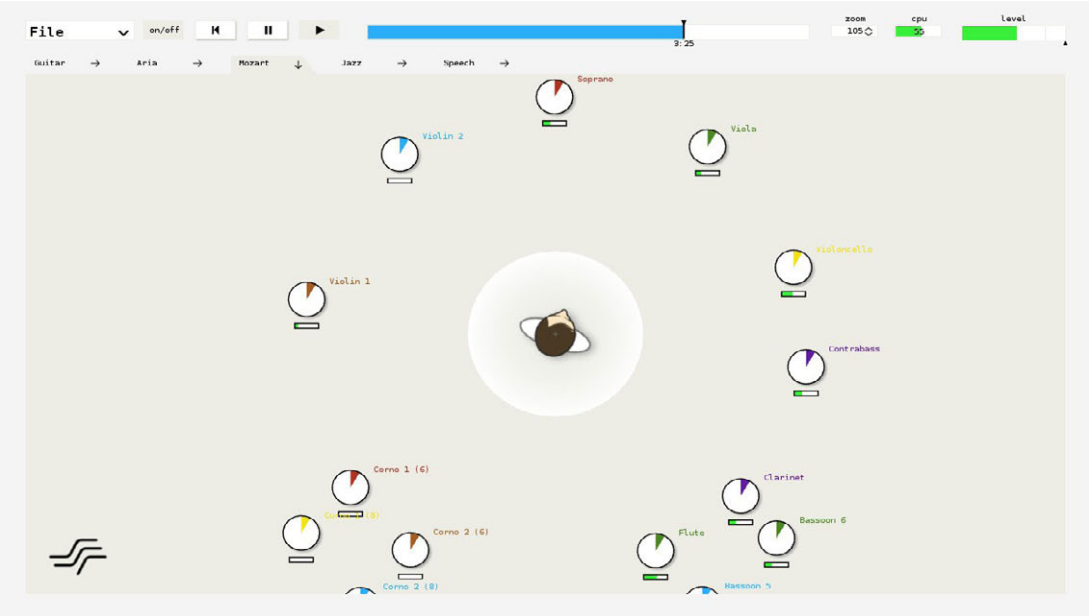

Figure 1: Screen shot of the SSR GUI [12]

4.3 Why use SOFA

The SSR is intended as a tool for spatial audio research. For this purpose sometimes is necessary to have access to some different HRTFs. The format used in the SSR is based in wav files. Every HRIR is stored in a channel of the wav file, and then the SSR read them. Even though is possible to adapt different HRTFs to this format would be better to use a standard format.

SOFA fits in the needs of the binaural renderer from the SSR. Is intended to be a standard for storing HRTFs (among other acoustics measurements). As a standard format it supposed that the researching groups are going to use it to store their measurements.

Adding the capability to the SSR to read SOFA files increase the number of HRTFs that can be used with a minimum effort. Only is necessary to configure the SSR to change the file where the HRTFs are stored.

Other reason to use SOFA inside the SSR is that it allow to store 2D HRTFs but also 3D. Not all the databases supported by SOFA have 3D measurements, but some like ARI has. With this data, SSR can use even 3D measurements with the 2D GUI, only setting the elevation always to 0.

5. NetCDF description

SOFA uses the numeric container netCDF to store the data. NetCDF is a set of software libraries and data formats supporting the creation, access, and sharing of scientific data. It is self-describing, network-transparent, and machine-independent; it supports huge files, partial access within a file, and allows for data compression. NetCDF-4 is an improved version of the classic netCDF and is widely used in the field of climatology, meteorology, oceanography, and geographic information systems. It is based on the HDF5 (HDF5 Group), a more basic numeric container.

NetCDF provides some libraries to manipulate the files. There are available for several programming languages like C, C++, Java, Fortran 77 and Fortran 90.Recent versions of matlab also support natively netCDF. There are also an independent package for octave, called octCDF [13].

5.1 General description of NETCDF

NetCDF has 2 main modes of storing the data: Classic data model [14] and Enhanced data model [15]. SOFA uses the enhanced model to have more options and flexibility in the future, but actually uses mostly the Classic model parts.

Classic mode is composed by different elements such as dimensions, variables and attributes:

- Dimensions: The dimensions are just numbers used to specify the size of the data stored on Variables. An unlimited dimension can grow at any time. The number of unlimited dimensions is 1 per file in the Classic data model.
- Variables: The variables are usually matrix, with the size given by the dimensions. They can have up to 5 dimensions. They contain the actual numerical data. Variables should be one of these types: char, byte, short, int, float, double
- Attributes: The attributes are just small texts attached to the global file, or to some specific variables, that explain some important characteristics. Are single dimensional arrays.

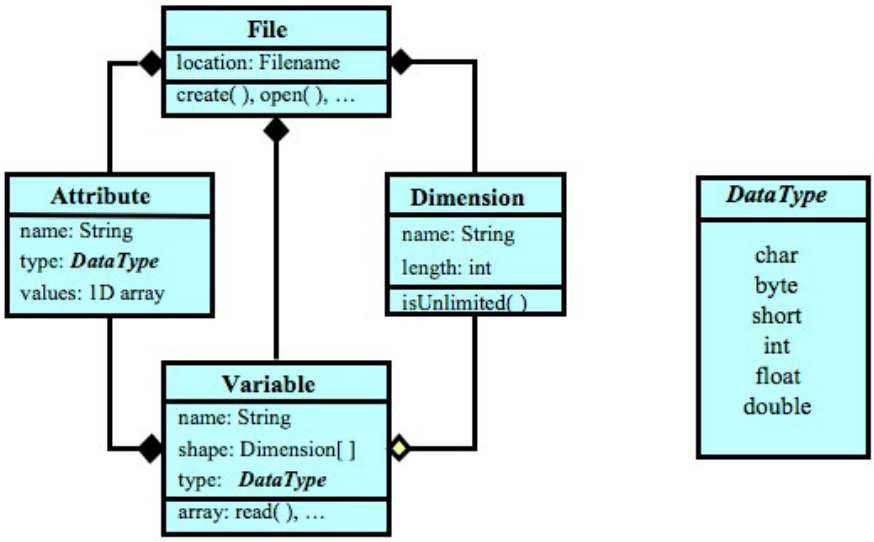

Figure 2: Classic model of netCDF [14]

5.2 netCDF4 and HDF5

As stated before SOFA uses the netCDF-4 enhanced data model. It is based in HDF5 (Hierarchical Data Format 5) but with some limitations. Enhanced data model contains the same elements that the Classic data model, but also adds 2 new elements from HDF5:

- Groups - A way of hierarchically organizing data, similar to directories in a Unix file system. A group may contain variables, dimensions, and attributes. In this way, a group acts as a container for the classic netCDF dataset. But netCDF-4/HDF5 files can have many groups, organized hierarchically.
- User-defined types - The user can now define compound types (like C structures), enumeration types, variable length arrays, and opaque types.

At the moment this features have not been added to SOFA specification but them allow more options in the future development of the format. Nowadays a SOFA file is a netCDF-4/HDF5 file with the limitations of the Classic data model, no support of multiple unlimited dimensions, groups and User-defined types.

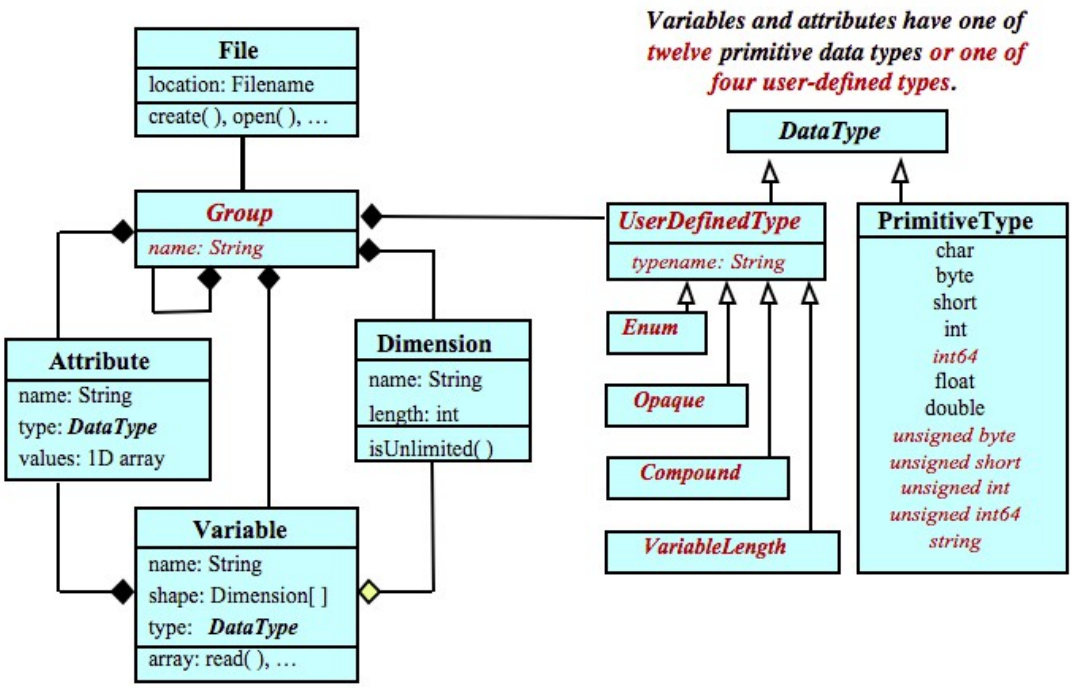

Figure 3: Enhanced Data model of netCDF4 [15]

6. SOFA description

6.1 SOFA Versions

During this project 2 different versions of SOFA have been made public with some differences between both.

6.2 First SOFA model

In order to describe the environment where the measurements were made, and some other useful information every SOFA file should have some compulsory dimensions, variables and attributes [16].

Name	Description
MDim (Measurements)	Number of measurements
RDim (Receivers)	Number of receivers
NDim (Samples)	Number of samples
CoordDim (Coordinates)	Used for defining coordinates, always 3
ScalarDim (Scalar)	Auxiliary dimension, always 1

Table 1: Basic dimensions of the first SOFA model

In the table above are listed the dimensions that should exist in every SOFA file. There are 2 different names, one is the name that suggest SOFA, and the other, between (), is the used in the real SOFA files provided by them.

Apart of the previous dimensions, some others are needed to create metadata variables. As a general rule the SOFA specifications says that a new dimension should be named with the name of the variable which needs the dimension plus 'DIM'. For example, if the variable 'RoomType" needs a dimension, it should be called 'RoomTypeDIM'. This is not used always, and sometimes the dimensions have the same name than the variables.

Once the dimensions have been defined the variables can be created. In this first SOFA version variables store data but also meta-data. In the case of HRTFs there is a data matrix that contains the impulse responses and then some other variables that contains information relative to the measurement.

The data variable is the most important in the SOFA file, but it needs all the information stored like metadata to be used properly. For complex measurements, netCDF allows to store data matrix as sub-fields of a bigger data matrix. For example, to store in the same SOFA file the impulse response and the frequency response of the same HRTF (data in time and in frequency respectively) it could be done storing the time data in Data.FIR and the frequency data in Data.Spectrum. This is not used in SOFA because matlab and octave have some problems handling them.

The metadata is usually related to the measurement setup, and also technical details like the SOFA version used, the measurement ID, audio latency, application for which the file is created, etc.

The most important metadata is the geometry description. This is necessary to know from which spatial point have been made the measurements, The distance, the angle, etc. In order to describe it, SOFA specifies some objects.

- Room: Describes the space where the measurements have been made. Can be 'free-field', 'shoe box' or 'collada'.
- Receiver (R): Any acoustical receptor like a microphone or an ear in the case of HRTFs.
- Listener (L): Is the group of all of the receivers. In the case of HRTFs would be lie the head of the person, but it also can be a microphone array for other kinds of measurements.
- Transmitter (T): A single emitter of sound. Can be a single driver of a loudspeaker.
- Source (S): The same as the Listener, but with the transmitters.

This objects should be located inside the whole measurement set-up. Usually each of them has a variable to store its position, its rotation, the direction to which is located, etc. Every of this fields needs also another variable that specifies in which format is these information. For the positions and the direction vectors, SOFA allows to store the coordinates in several coordinates systems like Cartesian, Spherical, Navigational, Vertical Polar and Horizontal Polar.
All of them have 3 coordinates, but with different meanings.

- Cartesian: [x y z]; all fields in meter.
- Spherical: [azimuth (deg, 0...360) elevation (deg, -90...+90); radius [m]].
- Navigational: [azimuth (deg, -180...+180) elevation (deg, -90...+90); radius [m]].
- Vertical Polar: [azimuth (deg, -90...+90) elevation (deg, -90...+270); radius [m]].
- Horizontal Polar: [lateral (deg, -90...+90) elevation (deg, -90...+270); radius [m]].

In order to make things clearer here is a real example of a SOFA file, generated with the scripts provided by the SOFA team for octave. It contains the data measured by ARI.
The name of the file is 'NH2.sofa'. It was generated with the octave script provided by SOFA team with some changes.

Name:	Value:
Scalar	1
Coordinates	3
Measurements	1550
Receivers	2
Samples	256
SchemeDIM	19
DataTypeDIM	3
SubjectIDDIM	3
DatabaseNameDIM	3
ApplicationNameDIM	21
ApplicationVersionDIM	5
TransmitterPositionTypeDIM	9
SourcePositionTypeDIM	9
SourceUpTypeDIM	9
SourceViewTypeDIM	9
ReceiverPositionTypeDIM	9
ListenerPositionTypeDIM	9
ListenerViewTypeDIM	9
ListenerUpTypeDIM	9
MeasurementIDDIM	4
RoomTypeDIM	10
SOFAVersionDIM	6

Table 2: Dimensions of the sofa file NH2.sofa

With these dimensions, all the variables can be defined. The next table shows the complete list of variables inside the NH2.sofa file. Is intended to show the structure of the dimensions and variables and their relation. To define a simple position of one of the objects of the set up is necessary to define first the dimension, after a variable to specify the coordinate system, and finally the position. For example, for the position of the source are needed the dimension SourcePositionTypeDIM, and the variables SourcePositionType and SourcePosition.

Name	Size	Type
Scheme	(SchemeDIM, Scalar)	char
Data.FIR	(Samples, Receivers, Measurements)	double
DataType	(DataTypeDIM, Scalar)	char
SamplingRate	(Scalar, Scalar)	float
SubjectID	(SubjectIDDIM, Scalar)	char
DatabaseName	(DatabaseNameDIM, Scalar)	char
ApplicationName	(ApplicationNameDIM, Scalar)	char
ApplicationVersion	(ApplicationVersionDIM, Scalar)	char
TransmitterPositionType	(TransmitterPositionTypeDIM, Scalar)	char
TransmitterPosition	(Coordinates, Scalar)	float
SourcePositionType	(SourcePositionTypeDIM, Scalar)	char
SourceUpType	(SourceUpTypeDIM, Scalar)	char
SourceViewType	(SourceViewTypeDIM, Scalar)	char
SourcePosition	(Coordinates, Scalar)	float
SourceView	(Coordinates, Scalar)	float
SourceUp	(Coordinates, Scalar)	float
ReceiverPositionType	(ReceiverPositionTypeDIM, Scalar)	char
ListenerPositionType	(ListenerPositionTypeDIM, Scalar)	char
ListenerViewType	(ListenerViewTypeDIM, Scalar)	char
ListenerUpType	(ListenerUpTypeDIM, Scalar)	char
ListenerPosition	(Coordinates, Scalar)	float
ListenerView	(Coordinates, Scalar)	float
ListenerUp	(Coordinates, Scalar)	float
ListenerRotation	(Coordinates, Measurements)	float
MeasurementID	(MeasurementIDDIM, Scalar)	char
MeasurementParameterSourceAudioChannel	(Scalar, Measurements)	float
MeasurementParameterItemIndex	(Scalar, Measurements)	float
MeasurementParameterSourceAmplitude	(Scalar, Measurements)	float
RoomType	(RoomTypeDIM, Scalar)	char
ReceiverPosition	(Coordinates, Scalar)	float
SourceRotation	(Coordinates, Scalar)	float
SOFAVersion	(SOFAVersionDIM, Scalar)	char

Table 3: Variables of the SOFA file NH2.sofa

At last here is a small description of some of the variables. Most of them are well described only with the name:

- Data.FIR: The actual data. The data could not be arranged by position, could be mixed. That means for example, that the first measurement could corresponds to the position 0º azimuth, 0º elevation and the next one to 50º azimuth, -10 elevation.
- DataType: String that specifies the kind of data stored in the data matrix.
- SamplingRate: Sampling rate of the data in Hz.
- SubjectID: Identification of the subject. It depends on the database used
- DatabaseName: Name of the database used. ARI, T-labs, CIPIC,... .
- 'Variable'PositionType: A string indicating the coordinates system used in the 'Variable'Position, it could be Cartesian, spherical, etc. 'Variable' can be any of the objects, Receiver, Listener, Transmitter or Source.
- 'Variable'Position: A matrix with the size X x 3, where X is the number of 'Variables' in the set up. It indicates the position using the coordinates system specified in 'Variable'PositionType.
- 'Variable'UpType: A string indicating the coordinates system used in the 'Variable'Up.
- 'Variable'Up: A vector that indicates the direction where the 'Variable's' top is pointing
- 'Variable'ViewType: A string indicating the coordinates system used in the 'Variable'View.
- 'Variable'View: A vector that indicates the direction where the 'Variable's' front is pointing.
- ListenerRotation: Important metadata for HRTFs. Is a 2 dimensional matrix with the size Measurements x Coordinates. It describes the angle between the Listener and the Source. The coordinates represent respectively the azimuth, the elevation and the roll of the Listener in relation to the Source.
- RoomType: Describes the space where the measurements have been made. Can be 'free-field', 'shoe box' or 'collada'.
- SOFAVersion: The number of SOFA version.

6.3 Second SOFA model

The SOFA version described above is the first version and is the used for the test and for designing the API in this project. There is a newer SOFA version [17] but without any example file, because of that the first one have been used with.

Here is a description of the new SOFA model to give a general idea of what SOFA is going to be in the future. Even though both version share the same main ideas, there are some differences.

The first change between versions is that in the new one, the Transmitter changes its name to a much clearer one, Emitter.

The way to define the spatial relation between objects also changes a bit, but still keeping the up and view vectors for each of the objects, and the coordinates system are the same. There is no a complete agreement about this in the new version, so it can change in the future versions.

NetCDF-4 with the Enhanced data model still being the numerical container used in the new version.

Regarding the data, the new version adds some others kinds of data that can be stored apart of impulse responses.

The main dimensions existing in the previous version still existing in the new one, and one new optional is added: Quaternions but no more information is given about it.

A good point of the new version is that some general metadata is now stored as attributes instead variables. With the attributes is not necessary to create a dimension with the size before creating the attribute and simplify the reading and writing. Also defines new metadata that can be included in the file. Some metadata can not be stored as an attribute, for example ListenerRotation. These kind of metadata is stored like the first version.

Some other kind of metadata changes from variables to attributes but instead attached to the global file, are attached to some other variable. Metadata of the type 'Variable'PositionType are in the new version attributes attached to 'Variable'Position.

A new concept is introduced, the idea of conventions. They are used normally in others formats that use netCDF4. A convention is a set of recommendations about the name of the variables and dimensions inside a netCDF4 file.

Two conventions are defined in the new SOFA version:

- Free-field HRTF measurement of a single listener (SimpleFreeFieldHRTF)
- Microphone-array measurement in a room (SimpleDRIRMicArray)

Only the first one is interesting for this project.

SimpleFreeFieldHRTF convention defines a set up with one Listener, a person, with 2 Receivers, the ears, and a single Source, an omnidirectional speaker. The RoomType should be free-field, and the measurements are done at a constant distance between Source and Listener. The measured HRTFs are stored as Finite Impulse Responses (FIR).

One difference between this convention and the previous SOFA format is that in the Data matrix the dimensions are changed. In the first version the size usually is [Samples, Receivers, Measurements] and in the new convention is [Measurements, Receivers, Samples]. Is not an important difference but should be noticed before working at the same time with both versions.

The different azimuths and elevations are stored in ListenerRotation, a variable with size [Measurements, Coordinates] with the coordinates system “din9300”, that means: azimuth, elevation, roll.

With all the stated before, it is easy to see that the differences between both versions are some but no so important. With not so much effort a system designed for working with the first version of SOFA can be adapted to work with the new version and the new convention.

7. Accomplished work

7.1 Why an API

The main goal of this project is to provide to the SSR the tools to make possible reading HRTFs stored with the SOFA format for using them in the binaural renderer. The best way to achieve it is to create a set of basic functions in C++ to manipulate SOFA and then integrate these functions to the current version of the SSR. In addition this API could be used for other purposes since the functions are intended to be general and not only oriented to the SSR.

7.2 API design

In order to make using this API easier, the functions should be designed to access the information at high level, that is that for reading and writing dimensions, for example, there is one function per every dimension and one general able to read or write any dimension. Low level functions currently exist in the API provided by netCDF for C++, so it has no sense to create new functions with the same functionality.

The idea for doing the API, is to create a class and write the necessary functions inside the class to make it useful.

One of the first ideas was to create a class with the same attributes (notice that now attributes is used as variables inside a C++ class, not like parts of netCDF) than a SOFA file, and then, read the requested attributes from that. The problem here is when only a certain data is needed, it has no sense to read all the data and metadata from the SOFA file and then read again the useful data. That was the reason because this option was discarded.

The second option was to create a class with a single attribute. An attribute that actually is an object from the class NcFile. Then create the methods using the API from netCDF for C++ to read only the useful information.

With the basic structure of the class defined, the next step is to define the necessary functions inside the class to be useful to read and write SOFA files.

The first necessary function in a class is the constructor. In this case, it should create and object from the class NcFile. For this is necessary to know the path of the SOFA file, if the goal is only read from it. But if the goal is create a new SOFA file, and write data on it, it is also necessary to specify the mode we want to open the file.

Since the file is ready to be read or written the methods to do it are the next step. Note that this API was designed for the first version of SOFA, so there are no methods to read or write netCDF attributes. All the methods here were designed to store all the metadata in variables.

The dimensions are the first elements that should be read before reading the actual data. For that reason are the first methods designed. The process of read/write a dimension is always the same, the only thing that changes is the name of the dimension. Therefore a general method are necessary to make this task easier.

One general function to read a dimension by name, and other function that creates a new dimension in the SOFA file with arbitrary name and value.

For the variables the idea is more or less the same but with one difference. Because the metadata is stored as netCDF variables there are 2 different kinds of variables. Variables that store numerical data, and variables that store strings.

For the variables that store numerical data the same pair of functions than for the dimensions were though, like because all of this kind of variable are from the type float or double, only this 2 options have been considered. For the strings variables only the function to read them.

There are basic functions to read and write dimensions and variables, the next step is to design the functions to read/write each of the dimensions and variables with more simple methods.

For each of the basic dimensions should be a pair of functions, one for reading and other for writing. The basic dimensions as stated before are:

- Samples
- Measurements

- Receivers
- Emitters (this one was added later)

For reading or writing other dimensions the previous functions should be used.

For variables is more or less the same, some variables have their own function for reading and others should use the generic. The method for writing variables is only one.

Something similar is the case of the strings stored in variables. The most important have functions, but other should use the generic method.

Other useful functions for making easier read the data were designed. Two of these functions are for get only one of the measurements of the data. The other was necessary when there is a given position in the space, defined by 2 angles, azimuth and elevation, and is necessary to know where the data relative to this position is stored. That is because the data variable is not arranged by angle.

Basically these are the basic functions of the API to make a first attempt to read and write SOFA files from C++.

7.3 API functions description

Here are the specifications of every function of the class. This is intended to be a reference for using the API.

7.3.1 Constructors

sofa(const std::string filepath)

Opens the file specified in filepath in only read mode.

sofa(const std::string filepath,netCDF::NcFile::FileMode mode)

Opens the file specified in filepath with some of this modes:

- netCDF::NcFile::new: The program will create a new file only if the file does not exist.
- netCDF::NcFile::replace: It will create an empty file even if the file already exist.
- netCDF::NcFile::read: Does not allow writing.
- netCDF::NcFile::write: Allows write in the file.

7.3.2 Dimensions

get_samples()

Description: Read The value of the dimension Samples.

Returns: Integer with the value of the dimension Samples.

set_samples(new_samples)

Description: Write the value new_samples in dimension Samples. Notice that can not overwrite if Samples already exists

Arguments:

- new_samples: New value for the dimension Samples.

Returns: Boolean. True if success writing, false if not.

get_measurements();

Description: Read The value of the dimension Measurements.

Returns: Integer with the value of the dimension Measurements.

set_measurements(new_measurements)

Description: Write the value new_measurements in dimension Measurements. Notice that can not overwrite if Measurements already exists

Arguments:

- new_measurements: New value for the dimension Measurements.

Returns: Boolean. True if success writing, false if not.

get_receivers()

Description: Read The value of the dimension Receivers.

Returns: Integer with the value of the dimension Receivers.

set_receivers(new_receivers);

Description: Write the value new_receivers in dimension Receivers. Notice that can not overwrite if Receivers already exists

Arguments:

- new_receivers: New value for the dimension receivers.

Returns: Boolean. True if success writing, false if not.

get_emitters()

Description: Read The value of the dimension Emitters

Returns: Integer with the value of the dimension Emitters.

set_emitters(new_emitters)

Description: Write the value new_emitters in dimension Emitters. Notice that can not overwrite if Emitters already exists

Arguments:

- new_emitters: New value for the dimension Emitters.

Returns: Boolean. True if success writing, false if not.

set_coordinates()

Description: Write the value 3 in dimension Coordinates. Notice that can not overwrite if Coordinates already exists

Returns: Boolean. True if success writing, false if not.

7.3.3 Variables

get_RoomType()

Description: Read the char array stored in the variable RoomType and convert it to a string.

Returns: C++ string with the variable RoomType.

get_SofaVersion()

Description: Read the char array stored in the variable SofaVersion and convert it to a string.

Returns: C++ string with the variable SofaVersion.

get_DataType()

Description: Read the char array stored in the variable DataType and convert it to a string.

Returns: C++ string with the variable DataType

get_SourcePositionType()

Description: Read the char array stored in the variable SourcePositionType and convert it to a string.

Returns: C++ string with the variable SourcePositionType.

get_SourceViewType()

Description: Read the char array stored in the variable SourceViewType and convert it to a string.

Returns: C++ string with the variable SourceViewType.

get_SourceUpType()

Description: Read the char array stored in the variable SourceUpType and convert it to a string.

Returns: C++ string with the variable SourceUpType.

get_TransmitterPositionType()

Description: Read the char array stored in the variable TransmitterPositionType and convert it to a string.

Returns: C++ string with the variable TransmitterPositionType.

get_ListenerPositionType()

Description: Read the char array stored in the variable ListenerPositionType and convert it to a string.

Returns: C++ string with the variable ListenerPositionType.

get_ListenerViewType()

Description: Read the char array stored in the variable ListenerViewType and convert it to a string.

Returns: C++ string with the variable ListenerViewType.

get_ListenerUpType()

Description: Read the char array stored in the variable ListenerUpType and convert it to a string.

Returns: C++ string with the variable ListenerUpType.

get_ReceiverPositionType()

Description: Read the char array stored in the variable ReceiverPositionType and convert it to a string.

Returns: C++ string with the variable ReceiverPositionType.

get_SourcePosition(position)

Description: Read the variable SourcePosition and store it in a float array. The user should create the array with the correct size beforehand.

Arguments:

- position: A pointer to the first element of the array where SourcePosition is going to be stored.

Returns: void.

get_SourceView(position)

Description: Read the variable SourceView and store it in a float array. The user should create the array with the correct size beforehand.

Arguments:

- position: A pointer to the first element of the array where SourceView is going to be stored.

Returns: void.

get_SourceUp(position)

Description: Read the variable SourceUp and store it in a float array. The user should create the array with the correct size beforehand.

Arguments:

- position: A pointer to the first element of the array where SourceUp is going to be stored.

Returns: void.

get_TransmitterPosition(position)

Description: Read the variable TransmitterPosition and store it in a float array. The user should create the array with the correct size beforehand.

Arguments:

- position: A pointer to the first element of the array where TransmitterPosition is going to be stored.

Returns: void.

get_ListenerPosition(position)

Description: Read the variable ListenerPosition and store it in a float array. The user should create the array with the correct size beforehand.

Arguments:

- position: A pointer to the first element of the array where ListenerPosition is going to be stored.

Returns: void.

get_ListenerView(position)

Description: Read the variable ListenerView and store it in a float array. The user should create the array with the correct size beforehand.

Arguments:

- position: A pointer to the first element of the array where ListenerView is going to be stored.

Returns: void.

get_ListenerUp(position)

Description: Read the variable ListenerUp and store it in a float array. The user should create the array with the correct size beforehand.

Arguments:

- position: A pointer to the first element of the array where ListenerUp is going to be stored.

Returns: void.

get_ReceiverPosition(position)

Description: Read the variable ReceiverPosition and store it in a float array. The user should create the array with the correct size beforehand.

Arguments:

- position: A pointer to the first element of the array where ReceiverPosition is going to be stored.

Returns: void.

get_Data(data)

Description: Read the variable Data.FIR and store it in a float array. The user should create the array with the correct size beforehand.

Arguments:

- data: A pointer to the first element of the array where Data.FIR is going to be stored.

Returns: void.

get_1_measurement(data, azimuth, elevation)

Description: Look for the closes measurement to the position given by azimuth and elevation. Then store it in data, which should be a float array created by the user with the size Samples x Receivers

Arguments:

- data: A pointer to the first element of the array where the measurement is going to be stored.
- azimuth: Float describing the azimuth of a spatial point. The closest measure to this point would be stored in data.
- elevation: Float describing the elevation of a spatial point. The closest measure to this point would be stored in data.

Returns: void.

get_1_measurement(data, azimuth, elevation, receiver)

Description: Look for the closes measurement to the position given by azimuth and elevation. Then store it in data, which should be a float array created by the user with the size Samples.

Arguments:

- data: A pointer to the first element of the array where the measurement is going to be stored.
- azimuth: Float describing the azimuth of a spatial point. The closest measure to this point would be stored in data.
- elevation: Float describing the elevation of a spatial point. The closest measure to this point would be stored in data.
- receiver: Integer. If 0 the data relative to the left receiver would be used. If 1 the data relative to the left receiver would be used.

Returns: void.

7.3.4 Auxiliary functions

get_index(float azimuth, float elevation)

Description: Read the variable ListenerRotation and looks for the measurement done at the closest point to the one defined by azimuth and elevation.

Arguments:

- azimuth: Float describing the azimuth of a spatial point.
- elevation: Float describing the elevation of a spatial point.

Return: An integer with the index of the closes measurement to the point defined by azimuth and elevation. Between 0 and Measurements-1

get_azimuth(index)

Description: read the variable ListenerRotation at the index [0][index].

Arguments:

- index: An integer that defines a measurement.

Return: A float with the azimuth of the measurement defined by index.

get_elevation(index)

Description: read the variable ListenerRotation at the index [1][index].

Arguments:

- index: An integer that defines a measurement.

Return: A float with the elevation of the measurement defined by index.

7.3.5 Low level functions

get_dim(dimname)

Description: Read The value of the dimension with the name dimname

Arguments:

- dimname: C++ string with the name of the dimension to read.

Returns: Integer with the value of the dimension with the name dimname.

set_dim(dimname, new_value)

Description: Write the value new_value in dimension named dimname. Notice that can not overwrite if dimname already exists.

Arguments:

- dimname: C++ string with the name of the dimension to write.
- new_value: New value for the dimension named dimname.

Returns: Boolean. True if success writing, false if not.

get_var(varname, var)

Description: Read the variable named varname and store it in a float array. The user should create the array with the correct size beforehand.

Arguments:

- varname: C++ string with the name of the variable to read.
- var: A pointer to the first element of the array where the variable varname is going to be stored.

Returns: void.

get_var(varname, var)

Description: Read the variable named varname and store it in a double array. The user should create the array with the correct size beforehand. It is an overloaded function.

Arguments:

- varname: C++ string with the name of the variable to read.
- var: A pointer to the first element of the array where the variable varname is going to be stored.

Returns: void.

set_var(varname, new_var, dims)

Description: Creates a new variable named varname, which has as dimensions those specified in the vector of NcDim named dims. The numerical data is stored in a matrix of the type float, the pointer of the first element of that matrix is new_var. This function has not been tested.

Arguments:

- varname: C++ string with the name of the new variable.
- new_var: pointer to the first element of an array that contains the numerical data. Its size should be the same that the specified by the dimensions stored in dims
- dims: C++ vector with NcDim as elements. It determine the size of the new variable.

Returns: void.

set_var(varname, new_var, dims)

Description: Creates a new variable named varname, which has as dimensions those specified in the vector of NcDim named dims. The numerical data is stored in a matrix of the type double, the pointer of the first element of that matrix is new_var. It is an overloaded function. This function has not been tested.

Arguments:

- varname: C++ string with the name of the new variable.
- new_var: pointer to the first element of an array that contains the numerical data. Its size should be the same that the specified by the dimensions stored in dims
- dims: C++ vector with NcDim as elements. It determine the size of the new variable.

Returns: void.

get_string(stringname)

Description: Read the char array stored in the variable named stringname and convert it to a string.

Returns: C++ string with the variable named stringname.

7.4 Limitations and possible improvements

The functions based in get_var are not optimal because the way to give back the information read from the SOFA file can be confusing. With a pointer to the first element is impossible to know the size of the data. That is why the user should create beforehand an array with correct dimensions to store there the data read from the SOFA file.

Other function that is not complete is get_index. Ideally it should considerate all the source and listener positions, views and up vectors, etc. With all that information and the relation between both objects the returned index would be the optimal. In this version of the function with the file NH2.sofa, it does not take into account that information and when, for example, the position 90º azimuth, 0º elevation is requested, it gives back the index when the listener is rotated 90 degrees to the left, so, the real position returned by the function is 270º azimuth, 0º elevation. This can be easily solved for one specific case like this one, but the best solution

would be the presented above, that the function makes the correct choice without any user help. With the elevation does not happen this.

One problem with all the functions is that they are no well coded to handle errors. When, for example, the function set_samples tries to write the dimension samples and it actually exists, the program ends abruptly with a netCDF error, when an internal error message of the Api would be much better.

7.5 How to use the API

In order to use the API, some libraries should be installed. The usual way of building netCDF requires the HDF5, zlib, and curl libraries. Versions required are at least HDF5 1.8.8, zlib 1.2.5, and curl 7.18.0 or later[18].

Once the libraries are installed the sofa class can be used just including the header file in the program that needs it.

7.6 Implementation inside the SSR

Some of these functions have been included in the SSR code to try to use it with a SOFA file.

The changes were only made in the binaural renderer code. The functions to read a sofa file works with not so many changes from the original code. The SSR can run with the HRTFs stored in a SOFA file. That was tried with SOFA file containing only 2D measurements, the name of the file is 'QU_KEMAR_anechoic_3m.sofa'. The problem was that the HRTFs inside this file are not arranged like the original HRTFs used originally by the SSR the sound does not seem to come from the direction that GUI shows.

The solution for this is to use the function get_index inside the SSR. But, as stated previously, this function is not optimal, that is why it has not been included already in the code.

8. Future work

8.1 Future work inside the API

Some function can be improved, for example get_1_measurement could be better if it was able to give back a group of measurements between 4, or 2 if the measurements are only in the horizontal plane, given angles. For example, with a range between -45º and 45º in azimuth, and only in the horizontal plane (0º elevation) the function should returns a set of measurements that are inside this spatial area.

The most obvious improvement that can be made to the API is to adapt it to the new SOFA model. Some new functions would be necessary, like get_att and set_att for reading the attributes that are mandatory in the new model.

8.2 Future work in the binaural renderer

Despite the class have been tested inside the SSR, it needs some improvements for working correctly. The most important one is the commented in 7.4, regarding to get_index function. With that function well designed, the SSR could use SOFA files without many problems.

9. Conclusions

SOFA is a very new format and because of that some things are changing quick. The API described in this project is actually almost obsolete because the new version changes some important things, but it can be adapted to the new version.

When SOFA achieves a stable version would be much easier work with it, because it is easier to develop something when it sure that than can be updated some time. If there are changes every some time, sometimes is difficult to adapt the previous work to the new changes.

References

[1] http://sourceforge.net/projects/sofacoustics/

[2] http://www.unidata.ucar.edu/software/netcdf/

[3] http://www.hdfgroup.org/HDF5/whatishdf5.html

[4] http://spatialaudio.net/ssr/

[5] http://interface.cipic.ucdavis.edu/sound/hrtf.html

[6] http://recherche.ircam.fr/equipes/salles/listen/

[7] https://dev.qu.tu-berlin.de/projects/measurements/wiki/Impulse_Response_Measurements

[8] http://marl.smusic.nyu.edu/projects/HRIRrepository

[9] http://www.sp.m.is.nagoya-u.ac.jp/HRTF/database.html

[10]http://www.kfs.oeaw.ac.at/index.php?option=com_content&view=article&id=608&Itemid=606&lang=en

[11] http://sourceforge.net/p/sofacoustics/code/HEAD/tree/

[12] J. Ahrens, M. Geier, S. Spors , “ Introduction to the SoundScape Renderer (SSR) ” , 2012

[13] http://modb.oce.ulg.ac.be/mediawiki/index.php/NetCDF_toolbox_for_Octave

[14] http://www.unidata.ucar.edu/software/netcdf/docs/classic_model.html

[15] http://www.unidata.ucar.edu/software/netcdf/docs/enhanced_model.html

[16] http://sourceforge.net/projects/sofacoustics/files/SOFA%200.0.1%20%28deprecated%29.zip/download

[17] http://sourceforge.net/projects/sofacoustics/files/SOFA%20Specifications%20AES2013/download

[18] http://www.unidata.ucar.edu/software/netcdf/docs/build_default.html